Preserving Contextual Information in Relational Matrix Operations

Dolmatova, Oksana ; Augsten, Nikolaus ; Böhlen, Michael Hanspeter

Abstract: There exist large amounts of numerical data that are stored in databases and must be analyzed. Database tables come with a schema and include non-numerical attributes; this is crucial contextual information that is needed for interpreting the numerical values. We propose relational matrix operations that support the analysis of data stored in tables and that preserve contextual information. The result of our approach are precisely defined relational matrix operations and a system implementation in MonetDB that illustrates the seamless integration of relational matrix operations into a relational DBMS.

DOI: https://doi.org/10.1109/ICDE48307.2020.00197

Preserving Contextual Information in Relational Matrix Operations

Oksana Dolmatova
University of Zurich
Zurich, Switzerland
dolmatova@ifi.uzh.ch

Nikolaus Augsten
University of Salzburg
Salzburg, Austria
nikolaus.augsten@sbg.ac.at

Michael H. Böhlen
University of Zurich
Zurich, Switzerland
boehlen@ifi.uzh.ch

***Abstract*—There exist large amounts of numerical data that are stored in databases and must be analyzed. Database tables come with a schema and include non-numerical attributes; this is crucial contextual information that is needed for interpreting the numerical values. We propose relational matrix operations that support the analysis of data stored in tables and that preserve contextual information. The result of our approach are precisely defined relational matrix operations and a system implementation in MonetDB that illustrates the seamless integration of relational matrix operations into a relational DBMS.**

I. Introduction

Most data in relational databases includes numerical parts that must be analyzed. Since interesting analytical tasks often require both linear and relational operations, these operations must be combined to answer a single analytical query. However, linear operations are only defined and performed on numerical values and do not support the processing of the rich contextual information of relations. Our approach supports data analyses that require a mixture of relational and linear algebra operations.

Consider Figure 1 with relations s, f, and p from a plant bakery. Relation s records past, current, and planed future information about shift length (in hours) and number of workers per day. Relation f records information about the scrap percentage of different machines. Relation p records information about the profit from the sales of items. For instance, tuple s_1 states that on March 21, 2019, four workers were working a ten hour shift. Tuple f_1 states that, on this day, the scrap of mixing machine MM1 was 1.2%. Tuple p_1 states that on March 21, 2019, 53 items of Moscow bread were sold with a 5% rebate and a total profit of 6,100 rubles. Relation s is updated infrequently and includes planning information; relations f and p are updated regularly as new information about, respectively, scrap and profit becomes available.

As of January 2020, a revised work schedule with an increased number of workers and shorter shifts shall be put in place. Multiple linear regression [6], with independent variables L[1] and F, shall be used to predict the profit in 2020. To solve this type of analytical tasks we propose relational matrix operations that preserve contextual information. The preservation of contextual information is based on *splitting* relations into an *application part* (numerical data for matrix operations) and *contextual information* (giving meaning to the numerical data), which are processed independently during an operation, and *merging* the result of a linear operation with contextual information into the final result relation.

s (shift)

	Date	Length	Workers
	...	...	...
s_1	2019-03-21	10	4
s_2	2019-03-22	9	6
...	...	...	...
s_3	2020-01-01	5	15
	...	...	...

f (fault)

	Date	Machine	Fault
	...	...	...
f_1	2019-03-21	MM1	1.2
f_2	2019-03-21	BM1	0.3
f_3	2019-03-21	BM2	2.1
f_4	2019-03-22	BM1	2.5

p (profit)

	Date	Item	Rebate	Number	Profit
	...	...	...	...	...
p_1	2019-03-21	Moscow Bread	5	53	6,100
p_2	2019-03-21	Raisin Cake	7	70	11,900
p_3	2019-03-22	Salt Sticks	9	40	3,100
p_4	2019-03-22	Moscow Bread	2	120	12,200

Fig. 1: Sample database

Current relational systems are poorly equipped to solve complex analytical queries since not even basic matrix operations are natively supported. There have been efforts to integrate linear algebra into the relational model. Such approaches introduced matrices as ordered data structures that are either used as attributes in relations [1], [2], without the possibility to store contextual information for each matrix cell, or as standalone objects [5], [7], which offer only matrix operations without relational database features.

As a running example, we consider the three key steps of predicting the profit: (1) data preparation, (2) computing the linear regression, (3) predicting the future profit. The linear equation for the regression has the form $F * x_F + L * x_L = P$, where coefficients x_F and x_L denote the (possibly negative) contribution of F and L (independent variables) to profit P (dependent variable). Figure 2 illustrates the profit estimation expressed as a sequence of relational operations[2] and relational matrix operations (`cpd`, `inv`, `mmu` denote relational crossproduct, inversion, and matrix multiplication, respectively). Note the seamless integration of matrix and relational algebra op-

[1]We use the first character of the attribute name to refer to attributes.

[2]We write $X \rightarrow A$ to rename attribute/expression X to A.

erators: the entire process frequently switches between linear and relational operations.

$$
\left.\begin{array}{l} u1 \leftarrow {}_{D}\vartheta_{AVG(F)\rightarrow F}(f) \bowtie \pi_{D,L}(s) \\ u2 \leftarrow {}_{D}\vartheta_{SUM(P)\rightarrow P}(p) \end{array}\right\} \textit{prepare data}
$$

$$
\left.\begin{array}{l} u3 \leftarrow {}_{D;D}\mathtt{cpd}^{V}_{F,L;F,L}(u1,u1) \\ u4 \leftarrow {}_{V}\mathtt{inv}_{F,L}(u3) \\ u5 \leftarrow {}_{D;D}\mathtt{cpd}^{V}_{F,L;P}(u1,u2) \\ u6 \leftarrow {}_{V;V}\mathtt{mmu}_{F,L;P}(u4,u5) \end{array}\right\} \textit{compute linear regression}
$$

$$
\left.\begin{array}{l} u7 \leftarrow \sigma_{Year(D)=2020}(s) \times \vartheta_{AVG(F)\rightarrow F}(f) \\ u8 \leftarrow {}_{D;V}\mathtt{mmu}_{F,L;P}(u7,u6) \\ u9 \leftarrow {}_{Month(D)}\vartheta_{SUM(P)\rightarrow P}(u8) \end{array}\right\} \textit{predict profit}
$$

Fig. 2: Algebra expression for profit estimation in 2020

The steps to estimate the profit are as follows: 1) Data preparation ($u1$, $u2$): Relational operations are used to compute the average daily scrap ($u1$) and the daily profit per shift ($u2$). 2) Linear regression ($u3$, $u4$, $u5$, $u6$): We use the ordinary least squares (OLS) method [4, p. 25] to compute the linear regression. Thus, we compute $(\mathtt{CPD}(A,A))^{-1} * \mathtt{CPD}(A,V)$, where CPD denotes the matrix crossproduct. A is the matrix with the independent variables (i.e., $u1$), and V is the vector with the dependent variable (i.e., $u2$). 3) Profit estimation ($u7$, $u8$, $u9$): The coefficients defined by the linear regression are multiplied with the values of the corresponding independent variables (i.e., L and F) in 2020 to predict the future profit.

$u3$

V	F	L
F	7.8	34.5
L	34.5	181

$u4$

V	F	L
F	0.9	-0.17
L	-0.17	0.04

$u5$

V	P
F	59,850
L	317,700

$u6$

	V	P
z_1	F	-144
z_2	L	2533.5

Fig. 3: Steps during the linear regression computation

Note that all operations operate on and return relations, and preserve contextual information. Consider crossproduct $u5 = {}_{D;D}\mathtt{cpd}^{V}_{F,L;P}(u1,u2)$ from Figure 2. The rows of $u1$ and $u2$ are ordered according to the values of attribute D. The crossproduct is computed between the matrix consisting of the values of $u1.F$, $u1.L$ and the matrix of the values of $u2.P$, and the result relation includes an attribute V with contextual information. The result of the crossproduct is relation $u5$ with schema (V,P). The values of V are the attribute names of the application part of $u2$. These values are essential to interpret the tuples in $u6$. For example, tuple z_1 states that the profit decreases by 144 rubles for each percent of scrap (the independent variable F).

Our approach is purely based on relations and does not introduce any ordered data structures. Instead, the relevant row order for matrix operations is computed from *contextual information* in the input relations. At the system level we evaluate our approach by integrating it into a column store. We extended the kernel of MonetDB with relational matrix operations implemented over binary association tables (BATs). The column nature of MonetDB supports the splitting of a relation into application and non-application part, and the merging of a linear algebra result with contextual information into a result relation.

Our contributions are as follows:

- We propose new *relational matrix operations that preserve contextual information*. This is the first approach that performs relational matrix operations and does not require ordered data structures.
- The relational matrix operations are *closed with respect to the relational model*, i.e., all operations are applied to relations and return relations as results.
- We show that our solution is practically feasible and able to *leverage existing data structures* by integrating it into MonetDB.

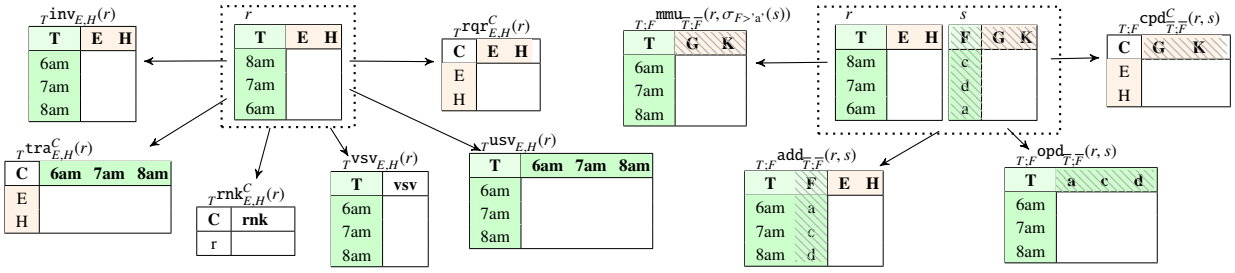
II. Context Preserving Relational Matrix Operations

Notation: A relation r is a set of tuples r_i with schema $\mathcal{R}$. A schema $\mathcal{R} = (A, B, \ldots)$ is a finite, ordered set of attribute names. Ordered subsets of a schema, $\mathbf{A} \subseteq \mathcal{R}$, are typeset in bold, and $\overline{\mathbf{A}} = \mathcal{R} \setminus \mathbf{A}$ denotes the complement of a set of attributes in a relation schema. An $n \times k$ matrix m is a two-dimensional array with n rows and k columns. The *column cast* maps a set of attribute values into an ordered set by ordering them; the column cast of attribute $\mathbf{O}$ is denoted by $\triangledown\mathbf{O}$. The *schema cast* $\Delta\mathbf{A}$ creates a matrix m (with a single column) from the attribute names of schema $\mathbf{A}$, preserving the attribute order. The result of *concatenating matrix* d and matrix e with k rows each is a new matrix $h = d \,\square\, e$ with k rows, where each row is the concatenation of the corresponding rows from d and e.

We consider the matrix operations from the R Matrix Algebra [3]: element-wise multiplication (EMU), matrix multiplication (MMU), outer product (OPD), cross product (CPD), matrix addition (ADD), matrix subtraction (SUB), transpose (TRA), solve equation (SOL), inversion (INV), eigenvectors (EVC), eigenvalues (EVL), QR decomposition (QQR, RQR), SVD – single value decomposition (DSV, USV, VSV), determinant (DET), rank (RNK), and Choleski factorization (CHF). Since QR and SVD return more than one matrix we split these operations (e.g., operations QQR and RQR for QR decomposition).

For each matrix operation we define how contextual information is preserved by the corresponding relational matrix operation. We use upper case for matrix operations (e.g., TRA) and lower case for relational matrix operations (e.g., tra). For each argument relation, r, of a relational matrix operation two parameters must be specified: (1) the *application schema* $\mathbf{A}$ identifies the attributes with the data to which the matrix operation is applied, (2) the *order specification* $\mathbf{O} \subseteq \mathcal{R} \setminus \mathbf{A}$ imposes an order on the tuples for this operation. The attributes of the order specification must form a key.

The application schema $\mathbf{A} \subseteq \mathcal{R}$ splits relation r into four non-overlapping areas: *application part* $\pi_{\mathbf{A}}(r)$; *non-application part* $\pi_{\overline{\mathbf{A}}}(r)$; application schema $\mathbf{A}$; and non-application schema $\overline{\mathbf{A}}$. The parts of r that do not include numeric matrix values,



(a) Unary Operations **(b)** Binary Operations

Fig. 4: Illustration of the preservation of contextual information

i.e., the schemas and the non-application part, form the *contextual information* of r. Intuitively, the application schema provides context for columns while the non-application part and schema provide context for rows.

Example 1: Application schema $\mathbf{A} = (E, H)$ splits relation r in Figure 4 into four parts: application part (white area); non-application part ($8am, 7am, 6am$); non-application schema (T); and application schema $\mathcal{R} = (E, H)$.

Matrix (μ) and relation (γ) constructors split and combine application part and context information to transition between matrices and relations without loosing relevant contextual information. At the implementation level, constructors are very efficient since they split and combine lists of attribute names and do not access the data (cf. Section III).

Figure 4 illustrates the preservation of contextual information for relational matrix operations. The cardinality of the results matrix determines the contextual information that is inherited. For instance, `inv` preserves row and column contextual information, whereas `cpd` transforms column contextual information of the first argument relation to row contextual information in the result.

Table I defines how input relations must be split and how result matrices are merged to relations. All definitions preserve contextual information as illustrated in Figure 4. Consider relational matrix inversion ${}_{\mathbf{O}}\mathtt{inv}_{\mathbf{A}}(r)$ with order specifications $\mathbf{O}$ and application schema $\mathbf{A}$. $\mu_{\mathbf{O};\mathbf{O}}(r)$ are the rows of the non-application part, $\mathtt{INV}(\mu_{\mathbf{O};\mathbf{A}}(r))$ is the result of matrix inversion, $\mathbf{O} \circ \mathbf{A}$ is the result schema.

TABLE I: Splitting and merging relations and matrices

${}_{\mathbf{O}}\mathtt{inv}_{\mathbf{A}}(r) = \gamma(\mu_{\mathbf{O};\mathbf{O}}(r) \square \mathtt{INV}(\mu_{\mathbf{O};\mathbf{A}}(r)), \mathbf{O} \circ \mathbf{A})$
${}_{\mathbf{O}}\mathtt{usv}_{\mathbf{A}}(r) = \gamma(\mu_{\mathbf{O};\mathbf{O}}(r) \square \mathtt{USV}(\mu_{\mathbf{O};\mathbf{A}}(r)), \mathbf{O} \circ \triangledown\mathbf{O})$
${}_{\mathbf{O}}\mathtt{vsv}_{\mathbf{A}}(r) = \gamma(\mu_{\mathbf{O};\mathbf{O}}(r) \square \mathtt{VSV}(\mu_{\mathbf{O};\mathbf{A}}(r)), \mathbf{O} \circ (\mathtt{op}))$
${}_{\mathbf{O};\mathbf{P}}\mathtt{add}_{\mathbf{A};\mathbf{B}}(r,s) = \gamma(\mu_{\mathbf{O};\mathbf{O}}(r) \square \mu_{\mathbf{P};\mathbf{P}}(s) \square \mathtt{ADD}(\mu_{\mathbf{O};\mathbf{A}}(r), \mu_{\mathbf{P};\mathbf{B}}(s)), \mathbf{O} \circ \mathbf{P} \circ \mathbf{A})$
${}_{\mathbf{O};\mathbf{P}}\mathtt{opd}_{\mathbf{A};\mathbf{B}}(r,s) = \gamma(\mu_{\mathbf{O};\mathbf{O}}(r) \square \mathtt{OPD}(\mu_{\mathbf{O};\mathbf{A}}(r), \mu_{\mathbf{P};\mathbf{B}}(s)), \mathbf{O} \circ \triangledown\mathbf{P})$
${}_{\mathbf{O};\mathbf{P}}\mathtt{mmu}_{\mathbf{A};\mathbf{B}}(r,s) = \gamma(\mu_{\mathbf{O};\mathbf{O}}(r) \square \mathtt{MMU}(\mu_{\mathbf{O};\mathbf{A}}(r), \mu_{\mathbf{P};\mathbf{B}}(s)), \mathbf{O} \circ \mathbf{B})$
${}_{\mathbf{O}}\mathtt{rnk}^{C}_{\mathbf{A}}(r) = \gamma(r \circ \mathtt{RNK}(\mu_{\mathbf{O};\mathbf{A}}(r)), (C, \mathtt{rnk}))$
${}_{\mathbf{O};\mathbf{P}}\mathtt{cpd}^{C}_{\mathbf{A};\mathbf{B}}(r,s) = \gamma(\Delta\mathbf{A} \square \mathtt{CPD}(\mu_{\mathbf{O};\mathbf{A}}(r), \mu_{\mathbf{P};\mathbf{B}}(s)), (C) \circ \mathbf{B})$
${}_{\mathbf{O}}\mathtt{rqr}^{C}_{\mathbf{A}}(r) = \gamma(\Delta\mathbf{A} \square \mathtt{RQR}(\mu_{\mathbf{O};\mathbf{A}}(r)), (C) \circ \mathbf{A})$
${}_{\mathbf{O}}\mathtt{tra}^{C}_{\mathbf{A}}(r) = \gamma(\Delta\mathbf{A} \square \mathtt{TRA}(\mu_{\mathbf{O};\mathbf{A}}(r)), (C) \circ \triangledown\mathbf{O})$

Operations with a different number of rows than any of the input relations add a new attribute C to the result relation with contextual information. Depending on the operation, the values of attribute C are the attribute names of the application schema of one of the input relations, or the name of the input relation.

III. Implementation and SQL Extension

A. MonetDB Integration

MonetDB stores each column of a table as a binary association table (BAT). A BAT is a table with a head and tail. The head is a column with object identifiers (OID), while the tail is a column with attribute values. B, X, Y denote lists of BATs and we use indices, e.g., Y_1, to refer to an individual BAT. All attribute values of a tuple in a relation have the same OID value. MonetDB operations manipulate BATs and all operations are represented and executed as sequences of BAT operations. An example BAT operation is $B_1 * B_2$, which is the element-wise multiplication. BAT operation $\downarrow$ sorts the OIDs of one BAT according to the order of the OIDs of another BAT from the same relation. For instance, $X \downarrow Y$ returns BAT X, whose OIDs have the same order as OIDs of BAT Y. $X \downarrow X$ denotes X sorted by its attribute values.

A relational matrix operation is processed in the following five steps: *1) Splitting* divides a relation into two parts; *2) Sorting* determines the order of the tuples for the matrix operation; *3) Morphing* transforms contextual information according to the operation; *4) Compute* performs the matrix operation on the values of the application part; *5) Merging* combines the result of the matrix operation with contextual information and constructs the result relation. Note that splitting and merging correspond to matrix and relation constructor, respectively, and work at schema level only.

Algorithm 1 illustrates the five steps for the relational matrix operation ${}_{\mathbf{O}}\mathtt{inv}_{\mathbf{A}}(r)$. Lines 2-6 correspond to two matrix constructors from the definition of ${}_{\mathbf{O}}\mathtt{inv}_{\mathbf{A}}(r)$ given in Table I: $\mu_{\mathbf{O};\mathbf{O}}(r)$ and $\mu_{\mathbf{O};\mathbf{A}}(r)$. The BATs of relation r are split, sorted, and morphed to get BATs X with the non-application part and BATs Y with the application part. Lines 7-17 illustrate the Gauss Jordan elimination for the $\mathtt{INV}(\mu_{\mathbf{O};\mathbf{A}}(r))$ computation. Function $IDmatrix(n)$ creates a list of BATs that represents an identity matrix of size $n \times n$. The selection operation $sel(B_1, i)$ returns the i^{th} value in B_1. In line 18, the relational constructor combines the result relation by merging the morphed non-application and the result application part.

```
WITH
  t1(F, D) AS ( SELECT AVG(F) AS F, D FROM fault GROUP BY D ),
  u1(D, F, L) AS ( SELECT t1.D, F, L FROM t1 NATURAL JOIN s )

SELECT SUM(P) AS P
FROM MMU ( SELECT D, F, L
           FROM shift, ( SELECT AVG(F) AS F FROM fault ) t2
           WHERE YEAR(D) = 2020
           ON F, L BY D,                                          u7
           SELECT *
           FROM MMU ( SELECT *
                      FROM INV ( SELECT *
                                 FROM CPD[V] (SELECT * FROM u1 ON F, L BY D,
                                              SELECT * FROM u1 ON F, L BY D)
                                 ON F, L BY V )
                      ON F, L BY V,                               u4
                      SELECT *
                      FROM CPD[V] ( SELECT * FROM u1 ON F, L BY D,
                                    SELECT D, SUM(P) AS P
                                    FROM profit
                                    GROUP BY D
                                    ON P BY D )
                      ON P BY D )                                 u5
           ON P BY V )                                            u6
GROUP BY MONTH(D);
```

Fig. 5: SQL statement that is equivalent to the algebra expression in Figure 2

The algorithm is representative for the other relational matrix operations and illustrates elegance and simplicity of the integration of our solution into MonetDB.

Algorithm 1: inv(**A**, **O**, r)

```
1  B ← BATs(r) ;
2  for b ∈ B do
3      if b ∈ A then C ← C ∪ b;                    // μ_{O;A}(r)
4      if b ∈ O then D ← D ∪ b;                    // μ_{O;O}(r)
5  X ← sort(D) ;
6  for b ∈ C do Y ← Y ∪ b↓X;
7  n ← Y.length ;                                  // INV(Y)
8  H ← IDmatrix(n);
9  for i = 1 to n do
10     v1 ← sel(Y_i, i);
11     Y_i ← Y_i/v1;
12     H_i ← H_i/v1;
13     for j = 1 to n do
14         if i ≠ j then
15             v2 ← sel(Y_j, i);
16             Y_j ← Y_j − Y_i * v2;
17             H_j ← H_j − H_i * v2;
18 Z ← Merge(X, H) ;                               // γ(X □ H, O ∘ A)
19 return Z;
```

B. SQL Extension for Relational Matrix Operations

The relational matrix operations have been made available in the from clause of SQL as, respectively, unary and binary operations with the names in Table I. Each argument relation r is defined through an extended SQL statement that allows to specify application schema and ordering as follows:

```
SELECT * FROM r ON A BY O
```

Figure 5 illustrates the syntax extension for the algebra expression in Figure 2. The relational matrix operations and the extension to specify application schema and ordering are highlighted in red.

IV. Summary and Future Work

In this paper, we proposed relational matrix operations that preserve contextual information. We described an integration of relational matrix operations into SQL and MonetDB that leverages the internal data structures of MonetDB and does not require changes in the query processing pipeline. Our approach excels for queries that combine relational and matrix algebras.

In terms of future work, the relational matrix operations open many opportunities for cross algebra optimizations. Thus, new query optimization techniques that optimize queries with both relational and linear algebra operations should be developed and implemented.

References

[1] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. The Multidimensional Database System RasDaMan. *SIGMOD Rec.*, 27(2), June 1998.

[2] S. Luo, Z. J. Gao, M. Gubanov, L. L. Perez, and C. Jermaine. Scalable linear algebra on a relational database system. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 523–534, April 2017.

[3] Quick R. R Matrix Algbera package overview. http://www.statmethods.net/advstats/matrix.html, 2017.

[4] C. Radhakrishna Rao and H. Toutenburg. *Linear Models, Least Squares and Alternatives*. Springer Series in Statistics. Springer-Verlag New York, 1995.

[5] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman. The Architecture of SciDB. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*, SSDBM'11, pages 1–16. Springer-Verlag, 2011.

[6] X. Yan and X. G. Su. *Linear Regression Analysis: Theory and Computing*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2009.

[7] Y. Zhang, M. Kersten, and S. Manegold. SciQL: Array Data Processing Inside an RDBMS. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1049–1052. ACM, 2013.